WHAT IS CLAIMED IS:

1. A method comprising:

dynamically binding an event context to an execution context in response to receiving an event by:

storing arriving events into a global event queue that is accessible by event contexts;

storing events from the global event queue in per-execution context event queues; and

associating event queues with the execution contexts to temporarily store the events for the event context for a duration of the binding to dynamically bind the events received on a per-event basis in the context queues.

2. The method of claim 1 wherein execution context can be in one of four states, idle, binding, bound, or unbinding.

3. The method of claim 1 wherein in the bound state, an execution context is bound to a specific event context and the execution context processes events for that event context and the event queue associated with that execution context is used to store events for the event context to which it is bound.

4. The method of claim 1 wherein in the unbinding state, the execution context determines if it has any more events to

process for the event context to which it was bound and either unbinds itself from the event context, going to idle state or begins processing another event from that context, going back to bound state.

5. The method of claim 1 wherein in the event context can be in one of two states, unbound or bound.

6. The method of claim 1 wherein the global FIFO event queue is used to queue events when the events first arrive into the system.

7. The method of claim 1 further comprising:

maintaining execution contexts in an idle state until an event arrives at a head of the global event queue.

8. The method of claim 1 wherein upon receiving an event, the method further comprises:

assigning an execution context that is in idle state to process the packet.

9. The method of claim 1 wherein binding an execution further comprises:

removing the event from the event queue;

determining a event context; and

determining if the event context to which this packet belongs is already bound to an execution context.

10. The method of claim 9 wherein if the event context is already bound, binding an execution further comprises

placing the packet in the event queue of the other execution context to which the event context associated with the packet is already bound to;

unbinding the event context; and

returning to an idle state.

11. The method of claim 10 wherein if the event context is not already bound, binding an execution further comprises

binding the execution context to that event context by updating a state of the execution context from idle to bound, updating the state of the event context from "not bound" to bound, and recording that this execution context is bound to this event context; and

processing the event.

12. The method of claim 11 wherein when the execution context completes processing an event, the execution context transitions to an unbinding state.

13. The method of claim 12 wherein when the execution context completes processing an event, the execution context checks its event queue for additional events to process;

14. The method of claim 12 wherein if there is at least one event in the queue, the execution context returns to the bound state, removes the packet from the queue and processes the packet, otherwise the execution context unbinds itself from the event context, and transitions to an idle state.

15. The method of claim 1 wherein the events are packets.

16. A method of processing data flows, the method comprising:

assigning a processing thread to an entry of a content addressable memory included in a processing engine of a processor including multiple processing engines;

receiving a data flow by the processor;

determining if an identifier associated with the data flow is stored in the entry in the content addressable memory; and

if the identifier is stored in the entry,

providing the received data flow to the thread for processing.

17. The method of claim 16 wherein the data flow includes a series of packets.

18. The method of claim 16 further comprising:

if it is determined that the entry is not stored in the content addressable memory,

determining if the identifier of the data flow is stored in a second entry included in the content addressable memory.

19. The method of claim 16 further comprising:

if it is determined that the entry is not stored, updating the entry to store the identifier of the data flow.

20. The method of claim 16 further comprising:

determining if the received data flow completes the data flow.

21. The method of claim 20 further comprising:

if the determined the data flow is complete, removing the identifier of the data flow from the entry.

22. A method comprising:

providing an execution thread a software token to place the thread into a free list, the token representing the position of the thread in a queue;

when the thread wakes up,

checking whether the thread has the token of the thread at the head of the queue.

23. The method of claim 22 wherein if the token matches then the thread is the correct thread and starts processing.

24. The method of claim 22 wherein if the tokens do not match, the method further comprises:

causing the current thread to go into a sleep state so that another thread can wake up.

25. The method of claim 22 wherein the tokens are derived from two pointers into an absolute general-purpose registers of a microengine of a processor having plural microengines.

26. The method of claim 22 wherein for received cells, cell order is maintained in processing the received cells.

27. A computer program product residing on a computer readable medium for dynamically binding an event context to an execution context in response to receiving an event comprising instructions for causing a processor to:

store events into a global event queue that is accessible by event contexts;

store events from the global event queue in per-execution context event queues; and

associate FIFO event queue with the execution context to temporarily store the events for that event context for a duration of the binding to dynamically bind the events received on a per-event basis in the context queues.

28. The computer program product of claim 27 further comprising instructions to:

maintain execution contexts in an idle state until an event arrives at a head of the global event queue.

29. The computer program product of claim 27 wherein upon receiving an event, the method further comprises instructions to:

assign an execution context that is in idle state to process the packet.

30. The computer program product of claim 27 wherein instructions to bind an execution further comprises instructions to:

remove the event from the event queue;

determine a event context; and

determine if the event context to which this packet belongs is already bound to an execution context.

31. The computer program product of claim 27 wherein if the event context is already bound, instructions to bind an execution further comprises instructions to:

place the packet in the event queue of the other execution context to which the event context associated with the packet is already bound to;

unbind the event context; and

return to an idle state.

32. The computer program product of claim 27 wherein if the event context is not already bound, instructions to bind an execution further comprises instructions to:

bind the execution context to that event context by updating a state of the execution context from idle to bound;

update the state of the event context from "not bound" to bound;

record that this execution context is bound to this event context; and

process the event.

33. A computer program product residing on a computer readable medium for processing data flows comprising instructions for causing a processor to:

assign a processing thread to an entry of a content addressable memory included in a processing engine of a processor including multiple processing engines;

receive a data flow by the processor;

determine if an identifier associated with the data flow is stored in the entry in the content addressable memory; and

if the identifier is stored in the entry,

provide the received data flow to the thread for processing.

34. The computer program product of claim 33 further comprising instructions to:

if it is determined that the entry is not stored in the content addressable memory,

determine if the identifier of the data flow is stored in a second entry included in the content addressable memory.

35. The computer program product of claim 33 further comprising instructions to:

if determined the entry is not stored, update the entry to store the identifier of the data flow.

36. The computer program product of claim 33 further comprising instructions to:

determine if the received data flow completes the data flow.

37. A computer program product residing on a computer readable medium for processing data flows comprising instructions for causing a processor to:

provide an execution thread a software token to place the thread into a free list, the token representing the position of the thread in a queue;

when the thread wakes up,

check whether the thread has the token of the thread at the head of the queue.

38. The computer program product of claim 38 wherein if the token matches then the thread is the correct thread and starts processing.

39. The computer program product of claim 38 wherein if the tokens do not match, the method further comprises:

causing the current thread to go into a sleep state so that another thread can wake up.

40. The computer program product of claim 38 wherein the tokens are derived from two pointers into an absolute general-purpose registers of a microengine of a processor having plural microengines.

41. The computer program product of claim 38 wherein for received cells, cell order is maintained in processing the received cells.

42. Apparatus comprising:

a processor including multiple processing engines, each processing engine including multiple event contexts;

circuitry to dynamically bind an event context to an execution context in response to receiving an event:

a global event queue that is accessible by all event contexts to store arriving events;

per-execution context event queues to store events from the global event queue; and

a FIFO event queue to temporarily store the events for that event context for a duration of the binding and to dynamically bind the events received on a per-event basis in the context queues.

43. The apparatus of claim 42 wherein the global FIFO event queue is used to queue events when the events first arrive into the system.

44. Apparatus for processing data flows, the apparatus comprising:

a processor including multiple processing engines; and

a memory executing a computer program product including instructions for causing the processor to:

assign a processing thread to an entry of a content addressable memory included in a processing engine of a processor including multiple processing engines;

receive a data flow by the processor;

determine if an identifier associated with the data flow is stored in the entry in the content addressable memory; and

if the identifier is stored in the entry,

provide the received data flow to the thread for processing.

45. The apparatus of claim 44 wherein the data flow includes a series of packets.

46. The apparatus of claim 44 wherein the computer program product further comprising instructions to:

determine if the identifier of the data flow is stored in a second entry included in the content addressable memory if it is determined that the entry is not stored in the content addressable memory.

47. The apparatus of claim 44 wherein the computer program product further comprising instructions to:

update the entry to store the identifier of the data flow if the entry is not stored.

48. A system comprising:

a router including an input port for receiving data packets and a switch fabric for determining the destination of the data packets; and

a processor capable of,

providing an execution thread a software token to place the thread into a free list, the token representing the position of the thread in a queue;

when the thread wakes up,

checking whether the thread has the token of the thread at the head of the queue.

49. The system of claim 48 wherein if the token matches then the thread is the correct thread and starts processing.

50. The system of claim 48 wherein if the tokens do not match, the processor is capable of:

causing the current thread to go into a sleep state so that another thread can wake up.

51. The system of claim 48 wherein the tokens are derived from two pointers into an absolute general-purpose registers of the processing engine of the processor.